Procedural Landscapes with Overhangs

Article · December 2003

Source: CiteSeer

CITATIONS
14

READS
318

2 authors, including:

Manuel N. Gamito
Framestore
28 PUBLICATIONS **252** CITATIONS

SEE PROFILE

Procedural Landscapes with Overhangs

Manuel N. Gamito
ADETTI
Edifício ISCTE
Av. das Forças Armadas, 1600 Lisboa
mag@iscte.pt

F. Kenton Musgrave
Pandromeda.com
15724 Trapshire Ct.
Waterford, VA 20197-1002, USA
musgrave@pandromeda.com

Abstract

Overhangs have been a major stumbling block in the context of terrain synthesis models. These models resort invariably to a heightfield paradigm, which immediately precludes the existence of any type of overhang. This article presents a new technique for the generation of surfaces, with the ability to model overhangs in a procedural way. This technique can be used generally to model landscape elements, not only terrains but also the surface of the sea. The technique applies non-linear deformations to an initial heightfield surface. The deformations occur after the surface has been displaced along some specified vector field. The method is conceptually simple and enhances greatly the class of landscapes synthesized with procedural models.

Keywords

Surface overhangs, Procedural models, surface deformation, ray-tracing, adaptive level of detail.

1. INTRODUCTION

The representation of terrains with mathematical models began with the seminal work of Mandelbrot on fractional Brownian motion [Mandelbrot83]. Mandelbrot realized that the output of an fBm process was visually similar to the ragged outline of a mountain range. He then proceeded to visualize a wire-frame model of a terrain, using a two-dimensional fBm process.

Several models have since been proposed for the synthesis of digital terrains, always relying on the concepts of fBm for the basic framework. Fournier and co-workers developed a polygon subdivision model, causing a triangular mesh to converge to a true fBm surface [Fournier82]. Voss used spectral synthesis to generate his terrains in the Fourier domain, based on the fBm power spectrum distribution [Voss85]. Saupe used Perlin's turbulence function, knowing that it had fractal properties [Saupe89]. This was the first procedural approach to terrain synthesis. Musgrave then pointed out that fBm, being a statistically homogeneous process, had drawbacks for terrain generation. In a true digital terrain map, gravity and erosion play an important role. Valleys, for instance, are always much smoother than mountain peaks due to the downward transport of sediments. This kind of subtleties is impossible to generate with an fBm model. Musgrave proposed a multifractal model, where each scale of the Perlin's turbulence function was weighted by the previously accumulated terrain height [Musgrave89]. Although this multifractal function has not yet been studied for its statistical properties, it is clear that it generates terrains with a much greater degree of realism.

Despite all the advances in the terrain generation techniques, terrains have always been represented with a heightfield paradigm. A heightfield representation is both flexible and intuitive with the only important drawback that it cannot represent overhangs. Overhangs are an ubiquitous feature in many actual terrains and can be present in a wide range of scales, from canyons and wind eroded rock formations to microstructures on the surface of rocks. Perhaps, the only known example of overhangs in the literature is related to breaking waves. It appears in the context of wave interactions with shorelines, with the consequent swelling and breaking of the wave trains [Fournier86].

It is clear that overhangs are an important issue that must be tackled in the context of landscape synthesis. The need for new algorithms with the ability to incorporate overhangs has already been acknowledged in the literature [Musgrave94]. This paper presents an attempt to achieve such a goal in a way that is as generic as possible. The model, here presented, can be used to generate overhangs, both in the traditional form of terrains and also in the form of breaking waves, following the work of Fournier. The main idea behind the technique consists in warping an initial heightfield surface along a vector field. The structure of the paper is as follows: In section 2 we present the algorithm, using the common framework of terrain models. In session 3 we discuss rendering issues. In section 4 we present some results and adapt the model to breaking waves. Section 5 gives some final remarks.

2. TERRAIN WARPING

Traditionally, terrains have been modelled by height functions $h = f(x,y)$: $\mathbf{R}^2 \alpha \mathbf{R}$ that associate an altitude h to each point (x,y) on the plane. These heightfields are then stored, either in memory or in disk, as two-dimensional altitude grids $h_{i,j} = f(x_i,y_j)$ for an underlying subdivision of the plane into equally spaced samples. Figure 1 shows an example of one heightfield terrain and also one terrain that cannot be represented with this paradigm. The problem with terrains exhibiting overhangs is that for some (x,y) values there is more than one altitude value. It is clear that the problem will be solved if we can somehow represent the terrain by some parametric function $\mathbf{S}(u,v)$: $\mathbf{R}^2 \alpha \mathbf{R}^3$. What remains to be seen is how this function can be generated.

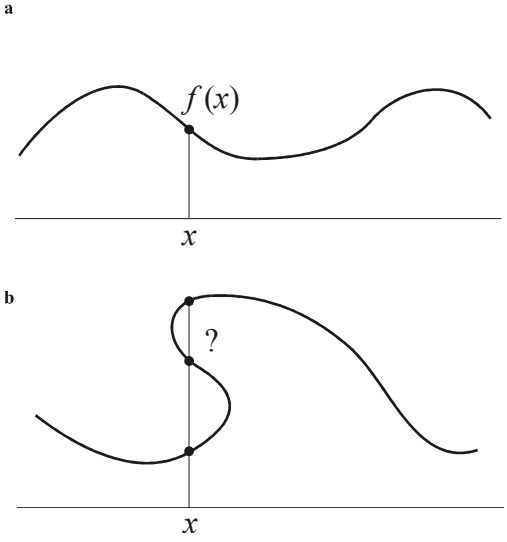

Figure 1. An heightfield terrain in (a) and a terrain with an overhang in (b).

Any algorithm that tries to generate non-heightfield terrains must obey two fundamental principles:

1. The terrain must be simply connected. This means that it must have a unique surface boundary separating an inside region from an outside region.
2. The terrain must not self-intersect. This constraint is obvious but is not that simple to implement.

The most naïve idea to implement non-heightfield terrains consists in the generation of a three-dimensional fractal field and the subsequent extraction of an isosurface from it. Voss used this method to obtain his images of "fractal flakes" [Voss85]. This technique violates condition 1 above, although it preserves condition 2. A more clever approach could use hypertextures as defined by Lewis and Perlin [Lewis89,Perlin89]. The problem with hyper textures is that the user must supply some initial volume density function. This function is subsequently "turbulated" and a fractal isosurface is extracted. It does not seem feasible to define explicitly a density function for every new terrain, unless some very specific geological feature is desired. Other hypotheses for terrain generation could include some simple deformation operations over an initial heightfield terrain (like deformation matrices or FFD lattices, for example) [Barr84,Sederberg86]. These techniques, however, are not general enough and have the significant drawback that condition 2 becomes very hard to hold. One would have to use some sort of collision detection algorithms to avoid self-intersections and thus control the maximum allowable amount of deformation.

A simple and elegant solution to this problem can be obtained by studying the apparently unrelated problem of fluid flow. Let's consider some flow field (a vector field $\mathbf{v}(\mathbf{x})$, expressing the velocity $\mathbf{v}$ at each point $\mathbf{x}$ in some n-dimensional space). Let's consider further a connected set of points inside the fluid at some initial time instant. This set can have a topological dimension of one (a line) or two (a surface). As time progresses, this set will be dragged by the surrounding fluid and will evolve into new shapes. This is usually called a *material set* since it is constituted by material particles that are dragged or *advected* by the flow [Batchelor67]. Figure 2 illustrates the idea. The two remarkable properties concerning the motion of material sets are that *they remain simply connected* if they were simply connected at the beginning and *they never self-intersect*, although different parts of the set can become arbitrarily close to each other. The conditions on the flow field for these two properties to hold are quite mild; all that is required is that the flow $\mathbf{v}(\mathbf{x})$ is continuous in the portions of space where the material set is supposed to evolve.

The advection of an initial surface through the action of some specified flow will be used to define the non-heightfield terrain. This operation will be called *terrain warping*, since it effectively warps the terrain from its initial shape to some new form. The initial shape of the terrain should be simple. It can be a heightfield terrain or even an horizontal plane. In the case of a horizontal plane, the flow field will be the sole responsible for adding shape and detail to the final surface.

The surface will be represented by a two-dimensional parametric function $\mathbf{S}(u,v,t)$. Every point on the surface is determined uniquely by a pair (u,v) of parameters. The variable t accounts for the evolution of the surface along the flow or, stated equivalently, the amount of warping that the surface has received.

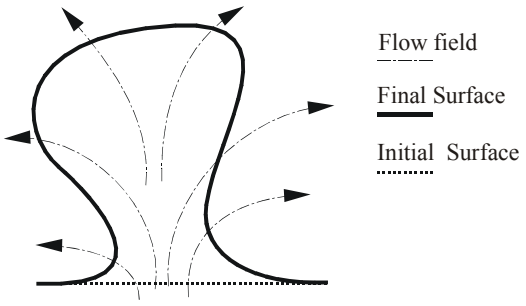

Figure 2. The motion of a material set of points under the action of a flow field.

Initially we will have $\mathbf{S}(u,v,0) = \mathbf{S}_0$, where $\mathbf{S}_0$ is the initial surface. Every point $\mathbf{x}(t) = \mathbf{S}(u,v,t)$ on the surface will then be advected along the flow, according to an ordinary differential equation of motion:

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}(\mathbf{x},t) \tag{1a}$$

$$\mathbf{x}(0) = \mathbf{S}_0(u,v) \tag{1b}$$

This equation means that each particle will have a velocity, at any given instant, equal to the velocity of the flow at the point where the particle lies. The particle will, therefore, describe a trajectory along the field lines of the flow. For generality, we are considering here flow fields $\mathbf{v}(\mathbf{x},t)$ that may be time dependent. We will see in section 4.2 an example where this time dependency of the flow becomes useful.

The surface advection is a point-wise operation, acting on each surface point, independently from all the others. This can, therefore, be considered as a procedural modelling technique. We can formally define a warping operator, denoted by $Warp\{.\}$, such that:

$$\mathbf{S} = Warp\{\mathbf{S}_0; \mathbf{v}; t\} \tag{2}$$

A useful mechanism that all procedural textures and geometries share is the built-in ability to provide adaptive level of detail and anti-aliasing. This is usually accomplished by band-limiting the frequency content of the object as a function of distance to the observer. The same mechanism can be applied here in several ways: either by band-limiting the spectral content of the initial surface $\mathbf{S}_0$, the spectral content of the flow field $\mathbf{v}(\mathbf{x})$, or even both. Both entities can be made to disregard high frequency components for points that are progressively farther away from the viewpoint, in a seamless manner. This is similar to the principle that is used with procedural heightfields and textures [Musgrave98].

The operator (2) is invertible because any given point $\mathbf{S}_0(u,v)$ maps to one and only one point in $\mathbf{S}(u,v,t)$ and vice-versa. The inverse can easily be shown to be:

$$\begin{aligned}\mathbf{S}_0 &= Warp^{-1}\{\mathbf{S}; \mathbf{v}; t\} = \\ &= Warp\{\mathbf{S}; \mathbf{v}; -t\}\end{aligned} \tag{3}$$

Notice that we only need to invert the time variable, causing the particles to flow backwards towards their initial positions. The inverse warping operator will be of particular importance when we discuss the topic of rendering in the next section.

A numerical integrator is necessary to implement the warping operator, expressed in equation (1), in a general way [Press92]. For efficiency reasons, however, one should always give preference to flow fields such that equation (1) has an analytical solution. This will greatly reduce the computational cost of the algorithm and also increase the accuracy of the final non-heightfield terrain.

Typical heightfield terrains can also be generated with the present technique. One simply needs to specify a deformation $\mathbf{v}(x,y) = (0,0,f(x,y))$ with a vertical component only, where $f(x,y)$ is the desired heightfield function, and apply it to a horizontal plane. This feature provides the added flexibility of generating both heightfield and non-heightfield terrains, in an integrated fashion, with the specification of appropriate deformation fields.

3. RENDERING NON-HEIGHTFIELD TERRAINS

The authors have found that a ray-tracing algorithm is the most convenient for the rendering of terrains with overhangs. The other alternative would be to tessellate the terrain, in the parameter space (u,v), into a triangular mesh and render it using any conventional rendering algorithm. This is not convenient, however, because the surface may be subject to arbitrarily high deformations. These would give rise to extremely stretched triangles that would no longer represent the terrain with sufficient accuracy. One would have to adaptively refine the tessellation as function of the local rate of deformation, using a technique similar to that of [Snyder92]. A direct ray-tracing approach obviates all these algorithmic complexities and accommodates naturally terrains with an infinite area of support.

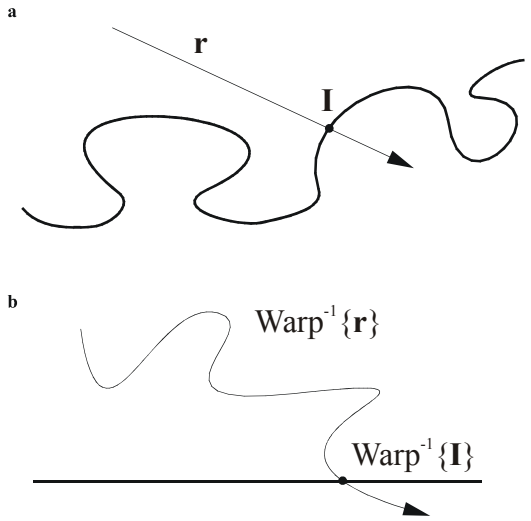

Figure 3. The intersection between a ray and the surface without inverse warping (a) and with inverse warping (b).

Two fundamental problems need to be solved in a ray-tracing context:

- Determination of the intersection point between a ray and the terrain.
- Determination of the terrain normal at the intersection point.

The first problem consists in determining, for any ray $\mathbf{r}(t)$ expressed in parametric form, the specific parameter $t = t_0$ for which the ray intersects the terrain. The intersection point is then given by $\mathbf{I} = \mathbf{r}(t_0)$. Instead of determining directly this ray-terrain intersection, we perform an inverse warping operation on the ray and determine its intersection with the initial surface. Figure 3 exemplifies this idea. The rationale is that the initial undeformed terrain has much less complexity than the non-heightfield terrain. It is better to find an intersection between a curve

and a simple shape than an intersection between a straight ray and a complex shape. The inverse warping operator will revert the terrain to its initial configuration, while transforming the ray into a parametric curve of arbitrary shape. In particular, if the initial terrain is the horizontal plane, the intersection test just needs to find the point where the vertical coordinate of the deformed ray changes sign. An intersection point, once found, must be warped back to its true position on the terrain. This type of ray domain deformation for the ray tracing of curved objects was initially proposed by Kajiya, although restricted to objects that were surfaces of revolution [Kajiya83].

3.1 Finding Terrain Intersections

The search for the intersection point between the deformed ray and the undeformed terrain is made with the help of an adaptive sampling of the ray, based on local curvature information. This idea is adapted from an algorithm described in [de Figueiredo95]. The algorithm recursively tests the smoothness of the ray between two points by sampling an intermediate point and finding the area of the enclosed triangle. In Figure 4, if point $\mathbf{P}_a$ is given by some parameter value t_a along the ray $\mathbf{r}(t)$, and point $\mathbf{P}_b$ by the parameter t_b, the intermediate point will be $\mathbf{P}_i = \mathbf{r}(0.5(t_a + t_b))$

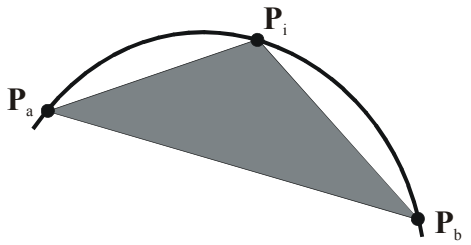

Figure 4. A subdivision of a parametric curve with local smoothness estimation based on the area of the triangle.

If the area is above a given threshold, the subdivision continues down to each of the two new intervals between the points, otherwise a straight line segment approximates that span of the ray. Every time such a straight segment is generated, it is also tested for intersection against the undeformed terrain. The following piece of pseudo-code clarifies the algorithm:

```
float CheckIntersection(Point Pa,
                        Point Pb) {
   float t;
   Point Pi = Midpoint(Pa,Pb);
   if (Area(Pa,Pi,Pb) < Epsilon)
       return Intersection(Pa,Pb);
   t = CheckIntersection(Pa,Pi);
   if (t > 0.0) return t;
   t = CheckIntersection(Pi,Pb);
   if (t > 0.0) return t;
   return -1.0;
}
```

In practice, one must also impose a maximum level of recursion to avoid locking the algorithm in sections where the ray might become discontinuous in the first derivative. Notice that, when descending to a lower level of subdivision, one always chooses first the interval that is closest to the eye point. In the case where the ray may intersect the terrain multiple times, this will guarantee that the first intersection found is the one which is closest to the viewer. We can also test for multiple intersections in the case where the terrain is transparent and light transport inside the medium must be considered. Again, the algorithm guarantees that the sequence of intersections will be sorted by increasing distance to the viewer, thereby facilitating the shading calculations.

The efficiency of the algorithm can be improved, at the cost of reducing somewhat its robustness, by noticing that parts of the ray far away from the surface need not be sampled with as much resolution. This effect is accomplished if we increase the tolerance factor for interval subdivision as a function of distance to the surface. The distance is taken to be the minimum distance of the two interval endpoints. In the simple case where the undeformed terrain is the horizontal plane, for instance, we can just replace the `Epsilon` factor in the previous pseudo-code by some linear function `Epsilon(z)` of the height z above the plane. The intersection routine will skim quickly through parts of the ray that are distant from the terrain and spend more time checking for intersections in parts that are at close proximity and where intersections are more likely to be found. This speedup, however, can lead to wrong intersections if the terrain exhibits very steep changes.

3.2 Finding Terrain Normal Vectors

Consider that an intersection was found at point $\mathbf{x}(u,v,T)$, where T is the amount of deformation and (u,v) are the parameter coordinates of the undeformed terrain. The normal at that point is calculated with the help of the two tangent vectors $\partial\mathbf{x}/\partial u$ and $\partial\mathbf{x}/\partial v$. These vectors can be computed by advecting them through the flow field, in exactly the same manner as the point $\mathbf{x}(u,v,T)$ itself was computed. For each of the two tangent vectors, we take the partial derivates at both sides of the advection equation (1). For instance, the trajectory of the tangent $\partial\mathbf{x}/\partial u$ will be given by:

$$\frac{d}{dt}\left(\frac{\partial \mathbf{x}}{\partial u}\right) = \nabla \mathbf{v} \cdot \frac{\partial \mathbf{x}}{\partial u} \qquad (4a)$$

$$\frac{\partial \mathbf{x}}{\partial u}(0) = \frac{\partial \mathbf{S}_0}{\partial u}(u,v) \qquad (4b)$$

where $\nabla\mathbf{v}$ is the gradient of the flow field. The tangent is initially equal to the tangent $\partial\mathbf{S}_0/\partial u$ of the undeformed terrain and evolves along with the flow up to the time T. The differential equation, giving the evolution of the tangent $\partial\mathbf{x}/\partial v$ is just like (4) if we replace the partials $\partial/\partial u$ by $\partial/\partial v$. If we consider again, the simplest case where the initial terrain is the horizontal plane, we will have:

$$\frac{\partial \mathbf{x}}{\partial u}(0) = (1,0,0) \quad (5a)$$

$$\frac{\partial \mathbf{x}}{\partial v}(0) = (0,1,0) \quad (5b)$$

Once an intersection point between a ray and the terrain is found, we warp it back to its position on the deformed terrain and, at the same time, obtain the two tangent vectors by solving the triple system of ODE's, consisting of the state vector $[\mathbf{x}\ \partial\mathbf{x}/\partial u\ \partial\mathbf{x}/\partial v]^T$, from $t = 0$ to $t = T$. With the two tangent vectors available, the normal is given by:

$$\mathbf{n} = \frac{\dfrac{\partial \mathbf{x}}{\partial u} \times \dfrac{\partial \mathbf{x}}{\partial v}}{\left\| \dfrac{\partial \mathbf{x}}{\partial u} \times \dfrac{\partial \mathbf{x}}{\partial v} \right\|} \quad (5)$$

The external product of the tangents gives the vector that is perpendicular to the terrain at the point of intersection. The division by the modulus insures the normal is a proper unit sized vector, which can be used for shading calculations.

The above technique for the computation of the two tangent vectors relies on a prior knowledge of the gradient vector $\nabla\mathbf{v}(\mathbf{x})$ of the flow field. We have come across situations where obtaining the gradient of the flow is either difficult to compute or generates expressions that are too complex. In this situation, it is possible to use a less accurate method, that relies solely on knowledge of the flow field $\mathbf{v}(\mathbf{x})$ itself, to estimate the two tangents by considering the central finite difference approximations:

$$\frac{\partial \mathbf{x}}{\partial u} = \frac{\mathbf{x}(u-\varepsilon, v, T) - \mathbf{x}(u+\varepsilon, v, T)}{2\varepsilon} \quad (6a)$$

$$\frac{\partial \mathbf{x}}{\partial v} = \frac{\mathbf{x}(u, v-\varepsilon, T) - \mathbf{x}(u, v+\varepsilon, T)}{2\varepsilon} \quad (6b)$$

for some suitably small value of ε. In practice, one should try successively smaller ε values and monitor the converge of the tangent vectors. More details of this technique can be found in [Press92].

It is also interesting to note that the terrain, being a two-dimensional parametric surface, can be subject straightforwardly to image mapping techniques. It is only necessary to establish a one-to-one correspondence between the pixel coordinates of the image texture and the pair (u,v) of parametric coordinates of the surface. This could be used to texture the terrain with areas of vegetation or snow, based on a previously calculated image map. We will give an example in the results section where this mapping has been used to advantage.

4. RESULTS

In this section, we present two examples of how the current surface deformation technique can be used to generate terrains with overhangs. We must point out beforehand that all models derived here do not attempt to follow any accurate physical laws. This is in accordance with the philosophy of procedural modelling techniques, where one tries to find simple mathematical functions to create simulations of natural phenomena that are visually convincing, even if they are not related to the underlying physics of the problem. This approach to the modelling of natural phenomena has been termed *ontogenetic modelling* in [Musgrave98]. It is, very often, a necessity due to the high complexity of the physical laws and is used in situations where one only wishes to compute animations that "look right", without special regard for the physical accuracy of the achieved results.

4.1 A Canyon Terrain

The first example consists of a procedurally generated canyon. The initial shape of the canyon is obtained by saturating a Perlin based noise function $n(x,y)$ to the discrete altitudes 0 and 1. The saturation is based on the sign of the noise function, i.e., negative areas of the noise become the bottom plane $z = 0$, while positive areas become the top plane $z = 1$. Once this starting shape is defined, we deform the vertical walls of the canyon according to a horizontal flow field (Figure 5a). The intensity of deformation is changed as a function of height, according to an fBm function. This allows us to model horizontal layers or strata of rock that are slightly displaced relative to each other, thus creating overhangs. It also incorporates adaptive level of detail by limiting the number of octaves of the vertical fBm function as a function of distance. The flow field itself is given by the gradient of the Perlin noise function: $\mathbf{v}(x,y) = \nabla n(x,y)$. Given the properties of the gradient operator and the fact that the lateral walls are described by the isoline $n(x,y) = 0$, this flow field guarantees that the displacement of the lateral walls is always performed along a perpendicular direction (Figure 5b).

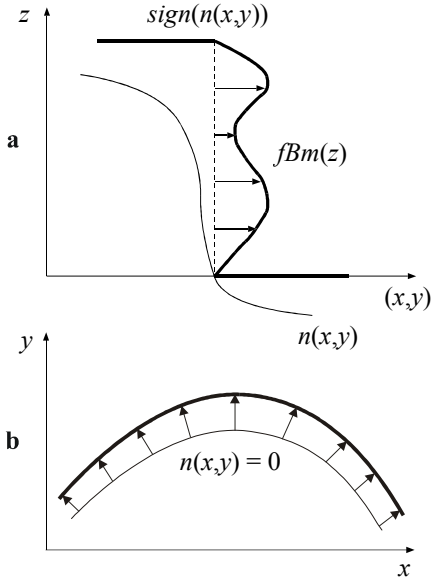

Figure 5. Construction of a canyon-like formation. Side view in (a) and top view in (b).

For this terrain configuration, the ray intersection test checks if each linear segment of the ray crosses from a negative to a positive area of the noise function, after the inverse warping operator has been applied. Consider

some portion of the ray $\mathbf{r}(t)$ from $t = t_a$ to t_b, which, according to our ray sampling algorithm, is small enough to be considered a straight segment. If it happens that $n(\mathbf{r}(t_a)) \times n(\mathbf{r}(t_b)) < 0$ then there must be an intersection for some parameter t_0 such that $t_a < t_0 < t_b$. The intersection parameter will be the solution of:

$$n(\mathbf{r}(t_0)) = 0 \qquad (7)$$

This is a non-linear equation, in terms of t_0, and since we know this unknown to be somewhere inside the interval $[t_a,t_b]$, we can apply a straightforward Newton-Raphson root-finder to converge to its true value. Once t_0 is known, we compute the intersection point $\mathbf{I}$ by warping back to the correct position in the deformed terrain:

$$\mathbf{I} = Warp\{\mathbf{r}(t_0); \nabla n; T\} \qquad (8)$$

Figure 6 shows the canyon with only one octave of an fBm function to deform the walls. The floor has not been rendered. The ability of procedural models, and of our non-heightfield procedural model in particular, to render terrains with infinite support becomes evident in this figure. Figure 7 shows the same canyon with eight octaves of fBm, deforming the walls at close range. In this case, the lower portion of the walls was sloped linearly with height to model a layer of weathering caused by the deposit of sediments from the upper strata. The floor of the canyon was also superimposed as a simple heightfield procedural terrain. This picture took 12 minutes of CPU time to render on a SGI O2 workstation at a resolution of 360x260 pixels.

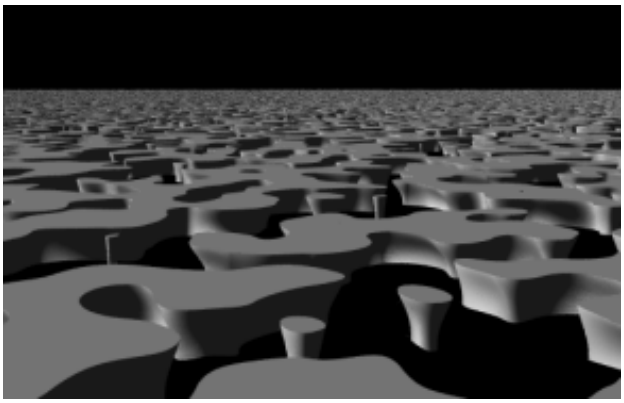

Figure 6. Saturation of a noise function into two discrete values with deformation of the lateral walls.

Figure 8 is another rendering of the same terrain from a different camera position. The rendering time for this picture was 1 hour of CPU time at a resolution of 360x360 pixels.

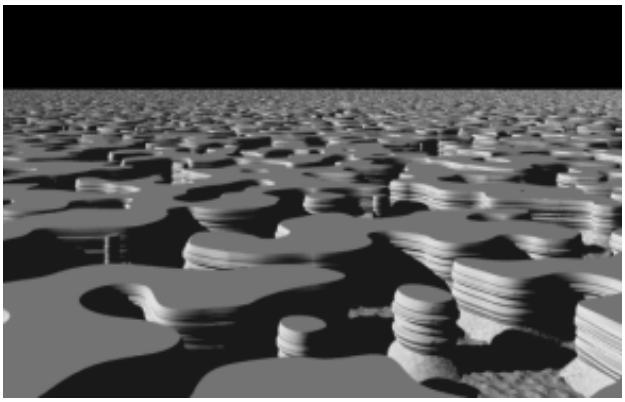

Figure 7. Same terrain as in Fig. 6 with increased deformation of the walls and a heightfield at the bottom.

The large disparity between these two computation times is related to the fact that the camera in Figure 8 is placed inside the canyon. For Figure 7, the camera is outside the canyon and each ray can be clipped against the range $0 < z < 1$ prior to any intersection checks because any intersection with the lateral walls must occur inside that range. This means, in practice, that only a relatively small portion of each ray must actually be considered for intersection. For Figure 8, on the other hand, this kind of optimisation cannot be applied and all of the ray must be checked for potential intersections.

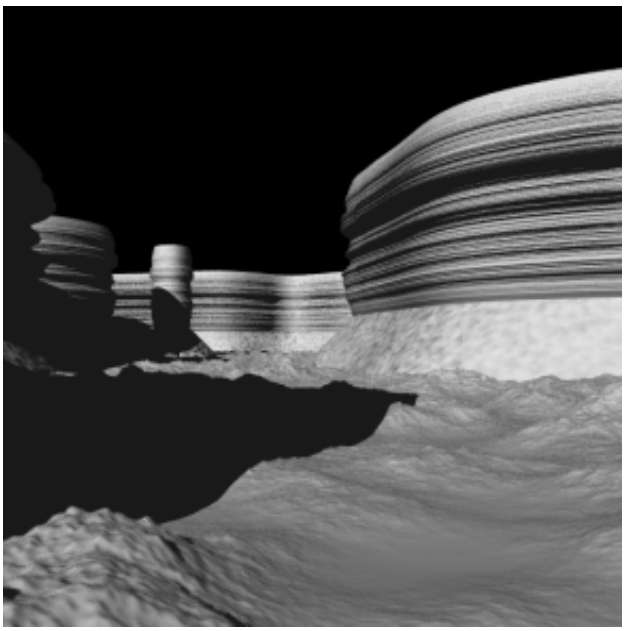

Figure 8. The terrain seen from a viewpoint inside the canyon.

4.2 Breaking Waves

In this example of a non-heightfield surface, we seek to reproduce and improve on some of the results given in [Fournier82]. Specifically, we seek to represent a breaking wave through all its stages of development. Other issues of the Fournier model, like the shallow water interaction with nearby shorelines, have been developed independently and are presented in [Gamito00]. We

intend to integrate the breaking wave model with our shallow water model in the near future.

The main restriction of the Fournier model, where breaking waves are concerned, is that, although the wave crest is deformed and an overhang is created, the wave crest never actually falls under gravity and hits the trough. To accomplish this effect, we take an horizontal plane to be the surface of the sea at rest and warp it along a deformation field with cylindrical shape. The expression $v(r)$ for the intensity of the deformation field is a function of distance r to the centre of rotation and is always tangential to the direction of rotation. Two radii of influence R_1 and R_2 and a decay factor α specify the field, according to the expression:

$$v(r) = \begin{cases} \left(1 - \dfrac{r - R_2}{R_1 - R_2}\right)^{\alpha} & \text{for } R_2 < r < R_1, \\ 0 & \text{otherwise} \end{cases} \qquad (9)$$

The deformation is more intense for $r = R_2$ and decays smoothly to zero at $r = R_1$. Figure 9 shows a sequence of deformations of the plane for increasing intensities of deformation. The parameters used were $R_1 = 0.75$, $R_2 = 20.0$ and $\alpha = 4.0$. To simulate the falling of the wave crest under gravity we use a secondary parabolic flow that implements correctly the law of gravity. The flow of equation (9) does not work for that purpose, as can be seen in Figure 10, because it would curl the wave into an ever increasing spiral.

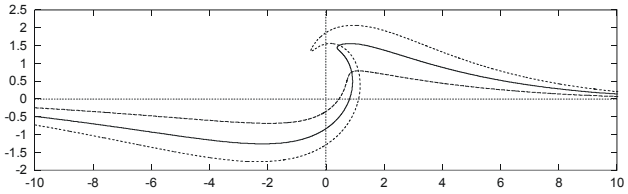

Figure 9. Several wave profiles for increasing intensities of deformation.

Any given particle switches from the circular to the parabolic flow once it crosses the half-line $\{x = 0; z > 0\}$, where coordinates are again relative to the centre of rotation of the flow. The parabolic flow is built such that every particle exhibits first and second order continuity of its trajectory at the transition point relative to the original circular flow. In this way, the deformed surface remains smooth everywhere, even though two different flow fields are being used to generate it. The expression for the parabolic flow is:

$$\mathbf{v}(x, z, t) = \left(-v, -\frac{v^2 t}{r}\right) \qquad (10)$$

for a wave that is travelling from the right to the left. In this equation, v is the horizontal velocity that each particle was carrying from the initial flow (9) at the transition point and $r^2 = (x^2 + y^2)$ is the distance to the centre of rotation at the same point.

4.2.1 Warping the surface of the wave

Say we want to warp some point $\mathbf{x}_0$ on the plane into some point $\mathbf{x}(T)$ on the wave for some amount T of deformation. This also means that we want to advect the point $\mathbf{x}_0$ along the flow for a time T. We start by applying the circular flow field (9) for the complete duration T:

$$\mathbf{x}(T) = Warp\{\mathbf{x}_0, \mathbf{v}_{circular}, T\} \qquad (11)$$

At the same time, we monitor the trajectory of the point and check if it starts a downward movement. This will happen, for some intermediate time $t < T$, when the point crosses the vertical axis above the centre of rotation. After that point, we begin a new deformation with the parabolic flow at the point where the previous deformation left off, i.e.:

$$\mathbf{x}(T) = Warp\{\mathbf{x}(t), \mathbf{v}_{parabolic}, T - t\} \qquad (12)$$

This advection is only applied for a time $T - t$, necessary to complete the total advection time of T from the initial point $\mathbf{x}_0$. Figure 10 shows the combined effect of the cylindrical and parabolic flows.

4.2.2 Inverse warping the surface of the wave

As we recall from section 3.1, an inverse warping operator must accompany every warping operator. This is so that we can check for ray-terrain intersections in undeformed space by applying the inverse deformation on the ray itself.

What we have been implementing in this particular case of a breaking wave is a form of state machine where each particle can undergo state changes when some predefined geometric conditions are verified. Different flow fields can then be applied based on the current state of each particle.

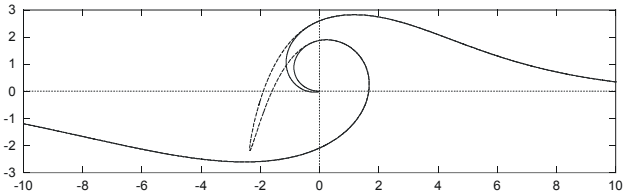

Figure 10. An excessive deformation can lead to incorrect results. A downward parabolic flow corrects the problem.

When we try to return from a point $\mathbf{x}(T)$, on the surface of the wave, to the original point $\mathbf{x}_0$ on the horizontal plane, two different hypotheses can happen:

1. The point is part of the plunging breaker and undergoes a state change from circular to parabolic flow.
2. The point is either in the rear or the trough of the wave and maintains a circular motion throughout the time T of the deformation.

If the point $\mathbf{x}(T)$ is on the rear of the wave (the region given by $x > 0$) then hypothesis 2 holds and we can simply apply:

$$\mathbf{x}_0 = Warp\{\mathbf{x}(T), \mathbf{v}_{circular}, -T\} \qquad (13)$$

If the point is in the front of the wave, the situation is a bit more ambiguous because it can either be part of the trough or the plunger (notice in Figure 10, in particular, that these two portions of the wave can easily intersect each other). We try hypothesis 1 first and apply:

$$\mathbf{x}_0 = Warp\{\mathbf{x}(T), \mathbf{v}_{parabolic}, -T\} \qquad (14)$$

always checking for the time $t < T$ when the particle switches to the circular flow. If we indeed find such a time, we return the correct point on the plane:

$$\mathbf{x}_0 = Warp\{\mathbf{x}(t), \mathbf{v}_{circular}, -(T-t)\} \qquad (15)$$

otherwise, the point is part of the trough and we can use equation (13) as it stands.

4.2.3 Animating the wave

It is quite straightforward to animate this wave model, if we just specify an intensity of deformation $T(y,t)$ that changes gradually along the axis y, around which the wave is curling, and also with time. Figure 11 shows several frames of the animation of a breaking wave that was produced with the above model. A short Mpeg clip of this animation can be found in [Gamito99].

There are several graphic elements in this animation that enhance the visual realism but are not directly related to the topic of this paper. In particular, the spray and the foam are modelled with procedural density functions. To render such elements, we march each ray forward in small increments and accumulate the opacity [Blasi93]. The backlighting on the surface of the wave is achieved by computing the distance travelled by each ray inside the wave and applying a simple exponential attenuation law. This means that both the entry point and the eventual exit point of each ray have to be computed. A procedural texture is also being applied on the top part of the plunging wave. As explained in section 3.2, this mapping can easily be accomplished since each point on the wave is identified by a unique (u,v) pair. This texture follows naturally the downward movement of the wave. The rest of the water surface is bump mapped with a three-dimensional procedural texture to simulate the small-scale ripples that are a characteristic of seawater.

Each frame of the animation took, on average, 40 minutes of CPU time for a resolution of 300x200 pixels. Despite the complexity of the scene present in these frames, the algorithm is quite efficient (compare with the computation time for Figure 8) because all the flow fields used in the breaking wave model have exact analytical solutions under the advection equation (1). This means that a numerical integrator for differential equations is not necessary and we can compute the $Warp\{\}$ operators with the simple evaluation of a mathematical expression. Some profiling tests, that we performed, have shown that only 40% of CPU time for the wave animation was spent in the ray-wave intersection tests. The remaining 60% of the computation time was spent integrating opacity values along the gas density functions and in computation of the atmospheric model.

5. CONCLUSIONS AND FUTURE WORK

The deformation of surfaces, driven by a vector field, provides a simple and extensible framework for the generation of overhanging structures. The operation is procedural, in the sense that it is performed point-wise and is controlled by a functionally defined vector field. It, therefore, inherits all the benefits of any procedural terrain modelling technique, like adaptive level of detail, built-in anti-aliasing and infinite support. The key to the success of this technique is to find appropriate deformation fields that will yield visually interesting results. This is in keeping with the philosophy behind procedural modelling techniques, where one tries to find simple mathematical functions that mimic natural phenomena to an acceptable degree. Two examples of surfaces with overhangs were already presented in this paper. We have seen, with these examples, how the basic technique can easily be adapted or extended to specific situations. More complex deformation fields should also be investigated, leading to more complex overhanging structures. The canyon terrain of section 4.1, for instance, could benefit if a small vertical deformation could also be added to the already existing horizontal deformation that generates the canyon walls. One other scenario, which is worth exploring in the future, is to apply the current deformation technique to surfaces that start out from an implicit representation, instead of the parametric representations that were used throughout this paper. Implicit surface representations can exhibit much greater topological richness that the relatively simple parametric representation. This added richness will certainly enlarge the range of non-heightfield surfaces that can be modelled with our technique.

The rendering of surfaces with overhangs is accomplished with a ray-domain inverse deformation. The deformed ray is sampled adaptively, based on its local curvature, and checked for intersection with the surface. The technique presented in this paper is not foolproof and can lead to wrong intersections whenever deformation fields with steep derivatives are present. The authors, however, have encountered no problems with the deformation fields used so far. We envisage that techniques like interval arithmetic or Lifchitz bounds could possibly be used to increase the robustness of the ray-surface intersection algorithm for any input deformation field [Snyder92,Hart96]. These robust techniques should also allow computational times to decrease significantly since larger steps along each ray can be taken with the confidence that no intersection will occur inside them.

6. REFERENCES

[Barr84] Barr, A.H., Global and local deformations of solid primitives, *in* H. Christiansen, ed., "Computer Graphics (SIGGRAPH '84 Proceedings)", vol. 18, pp. 21-30, 1984.

[Batchelor67] Batchelor, G.K., Fluid Dynamics, Cambridge University Press, ISBN 0-521-09817-3, 1967.

[Blasi93] Blasi, P., Le Saec, B., Schlick, C., A Rendering Algorithm for Discrete Volume Density Objects, *Computer Graphics Forum*, **12**(3), 201-210, 1993.

[de Figueiredo95] de Figueiredo, L.H., Adaptive sampling of parametric curves, *in* A. Paeth, ed., "Graphics Gems V", AP Professional, 1995.

[Fournier82] Fournier, A., Fussell, D., Carpenter, L., Computer rendering of stochastic models, *Communications of the ACM*, **25**(6), 371-384, 1982.

[Fournier86] Fournier, A., Reeves, W.T., A simple model of ocean waves, *in* D.C Evans & R.J. Athay, eds., "Computer Graphics (SIGGRAPH '86 Proceedings)", vol 20, pp371-384, 1986.

[Gamito99] Gamito, M.N., Musgrave, F.K., Non-heightfield rendering, <http://www.wizardnet.com/musgrave/sundry.html>

[Gamito00] Gamito, M.N., Musgrave, F.K., An accurate model of wave refraction over shallow water, *in* N. Magnenat-Thalmann, D. Thalmann, B. Arnaldi, eds., "Computer Animation and Simulation '2000", Eurographics, pp. 155-171, 2000.

[Hart96] Hart, J.C., Sphere Tracing: a geometric method for the antialiased ray tracing of implicit surfaces, *The Visual Computer*, **12**(9), 527-545, Springer-Verlag, 1996.

[Kajiya83] Kajiya, J.T., New techniques for ray tracing procedurally defined objects, *in* "Computer Graphics (SIGGRAPH '83 Proceedings)", vol. 17, pp. 91-102, 1983.

[Lewis89] Lewis, J.P., Algorithms for solid noise synthesis, *in* J. Lane, ed., "Computer Graphics (SIGRAPH '89 Proceedings)", vol 23, pp. 263-270, 1989.

[Mandelbrot83] Mandelbrot, B.B., The Fractal Geometry of Nature, W.H. Freeman and Co., New York, 1983.

[Musgrave89] Musgrave, F.K., Kolb, C.E., Mace, R.S., The synthesis and rendering of eroded fractal terrains, *in* J. Lane, ed., "Computer Graphics (SIGGRAPH '89 Proceedings)", vol. 23, pp. 41-50, 1989.

[Musgrave94] Musgrave, F.K., Texturing and Modelling: A Procedural Approach, 1st. ed., AP Professional, chapter 9, pp. 297-298, ISBN 0-12-228760-6, 1994.

[Musgrave98] Musgrave, F.K., Texturing and Modelling: A Procedural Approach, 2nd ed., AP Professional, ISBN 0-12-228730-4, 1998.

[Perlin89] Perlin, K., Hoffert, E.M., Hypertexture, *in* J. Lane, ed., "Computer Graphics (SIGGRAPH '89 Proceedings)", vol. 23, pp. 253-262, 1989.

[Press92] Press, W.H., Teukolsky, S.A, Vetterling, W.T., Flannery, B.P., Numerical Recipes in C: The Art of Scientific Computing, 2nd ed., Cambridge University Press, ISBN 0-521-43108-5, 1992.

[Saupe89] Saupe, D., Point evaluation of multi-variable random fractals, *in* J. Jurgens & D. Saupe, eds., "Visualisierung in Mathematik und Naturwissenschaft", Bremer Computergraphik Tage, Springer-Verlag, Berlin, 1989.

[Sederberg86] Sederberg, T.W., Parry, S.R., Free-form deformation of solid geometric models, *in* D.C. Evans & R.J., Athay, eds., "Computer Graphics (SIGGRAPH '86 Proceedings)", vol. 20, pp. 151-160, 1986.

[Snyder92] Snyder, J.M., Interval Analysis for Computer Graphics, *in* Catmull, E., ed., "Computer Graphics (SIGGRAPH '92 Proceedings)", vol. 26, pp. 121-130, 1992.

[Voss85] Voss, R.P., Fractal Forgeries, *in* R.A. Earnshaw, ed., "Fundamental Algorithms for Computer Graphics", Springer-Verlag, 1985.

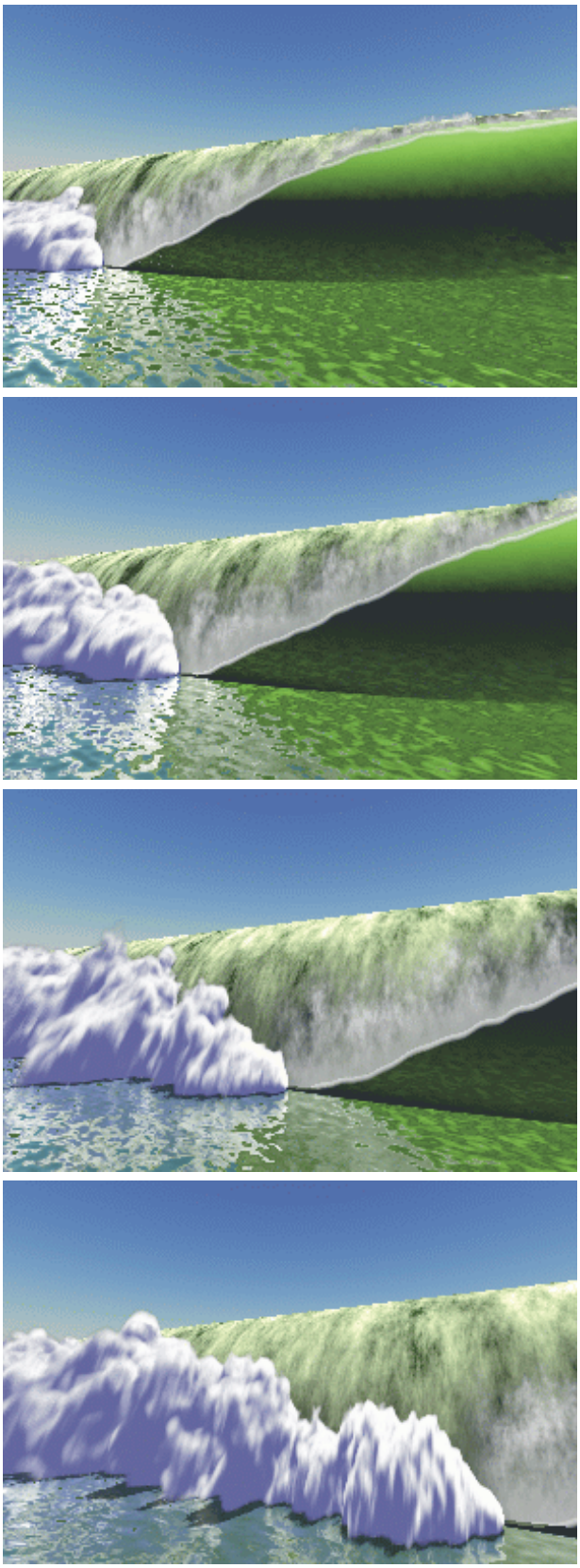

Figure 11. Several frames of the animation of a breaking wave. This is a complex scenario featuring non-heightfield waves, volumetric foam and spray, texture mapping, atmospheric light models, reflections and refractions on the surface of the water.